

Chapitre 2 :

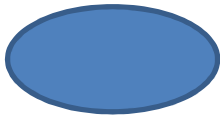
ALGORITHME

Algorithme

- **Définition:** un algorithme est une suite d'actions (opérations) à exécuter, pour l'obtention d'une solution (résultat) à un problème posé. Aussi, un algorithme est une description en langage naturel de la suite des actions effectuées par un programme structuré.
- **Action** : est une étape d'algorithme, c'est un évènement de durée déterminée.
 - Dans les exemples 1 et 2, chaque étape est une action.
 - Il est important de noter que : la séquence (l'ordre) d'actions doit être respectée (1, 2, 3, 4, 5).

Algorithme

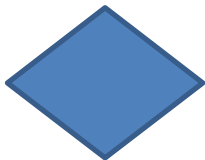
Organigramme : est une représentation graphique de l'enchaînement d'une suite d'actions. Un organigramme utilise des symboles graphiques :



le début et la fin d'un traitement.



Pour la majorité des actions.



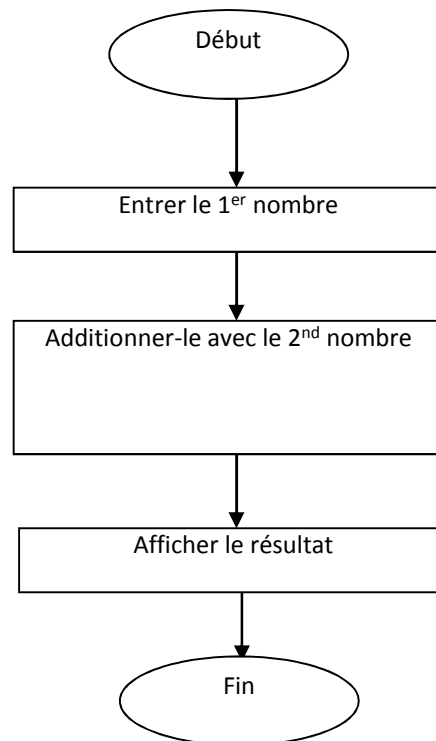
Pour l'action de test (condition).



Indique le sens de cheminement des actions.

Algorithme

La somme de 2 nombres à l'aide d'une calculatrice dans l'exemple 2 peut-être schématisée comme suit :



Algorithme

Propriétés d'un algorithme :

- Le concepteur d'un algorithme doit prévoir tous les cas possibles d'exécution.
- Un algorithme doit être toujours fini (nombre d'actions fini)
- La solution d'un même problème, peut être décrite avec plusieurs algorithmes (façons).
- Un algorithme est plus efficace qu'un autre, s'il est plus clair et plus rapide (moins d'actions)
- Pour la mise en pratique d'un algorithme sur l'ordinateur, il faut passer par un langage de programmation ou programme.

Algorithme

- Tout problème à programmer doit être résolu, d'abord sous forme d'algorithme, puis converti en programme dans le langage de votre choix.
- **Programme** : un programme informatique est une succession (suite) d'instructions (actions) exécutables par l'ordinateur. Ces instructions écrites dans un langage de programmation et traduisant un algorithme.

Algorithme

tous les programmes sont traduits en binaires (0 et 1) par un compilateur, pour qu'ils soient compréhensibles par la machine.

- **Compilateur** : est un programme qui permet de traduire un texte écrit dans un langage de programmation en langage machine (0,1).

Algorithme

- **Les étapes de résolution d'un problème :**

Pour résoudre un problème donné, on peut suivre les étapes suivantes :

- Analyser et comprendre l'énoncé du problème.
- Décomposer le problème en sous-problèmes plus simple à résoudre.
- Extraire à partir de l'énoncé du problème
 - Les données fournies, qui seront les « *entrées* ».
 - Les résultats désirés et recherchés par les problèmes, qui seront les «*sorties*».
 - Décrire les différentes étapes permettant de résoudre le problème, qui utilise les entrées, dans le but d'atteindre les sorties désirées.
- Elaboration d'un algorithme

Algorithme

La description de ces étapes donne ce qu'on appelle l'algorithme de résolution de ce problème, qui peut être décrit en langue naturelle ou dans un langage de description de l'algorithme.

Exemple 01:

Somme de
deux nombres
a et b.

Données: a, b

Résultat: somme

Calcul de la somme :

Somme=a+b

Algorithme

Exemple 02:

Moyenne de
trois nombres
x , y et z.

Données: x , y , z

Résultat: moyenne

Calcul de la moyenne :

$$\text{Moyenne} = (x+y+z)/3$$

Exemple 03:

Equation de
première
degré
 $ax+b=0$.

Données: a , b

Résultat: x

Calcul de la solution : $X = -b/a$

Algorithme

Exemple 04:
Equation de
deuxième degré
 $ax^2+bx+c=0$.

Données: a , b, c

Résultat: x1, x2

Calcul des solutions :

- $\Delta = b^2 - 4ac$
- Si $\Delta > 0$ il existe deux solutions : $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$ et $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$
- Si $\Delta < 0$ pas de solution
- Si $\Delta = 0$ il existe une solution unique : $x_1 = -b/2a$

LE LANGAGE ALGORITHMIQUE (LA)

Le Langage Algorithmique (LA)

1. Définition: l'algorithme est exprimé dans un langage naturel appelé **Langage Algorithmique (LA)** ou **pseudo-langage**, il décrit de manière structurée, claire et complète, les objets (données) manipulés par l'algorithme ainsi que l'ensemble des instructions à exécuter sur ces objets pour obtenir des résultats.

- un algorithme doit avoir
 - un nom
 - des déclarations
 - un groupe d'actions (corps de l'algorithme).

Algorithme

La forme générale d'un algorithme est la suivante :

Algorithme Nom de l'algorithme;

Déclaration

Constante < définition de constantes>

Type < définition de types>

Variable < définition de variables>

Début

<Partie Instructions> (Corps)

Fin

Algorithme

Un algorithme est une description finie d'un calcul qui associe un résultat à des données. Il est composé de 3 parties :

- **Entête** : qui spécifie le nom de l'algorithme par exemple :

Algorithme Somme ;

- **Déclaration** : contient la déclaration de différents objets (constants, variables, ..) utilisées dans l'algorithme.
- **Corps** : consiste en une suite d'opérations élémentaires (actions ou instructions). Ces différentes opérations sont délimitées par les mots clefs *Début* et *Fin*.

Algorithme

1.1. Partie déclaration

- Tous les objets qui rentrent en jeu pour l'exécution d'un algorithme doivent être déclarés, dans la partie déclaration, avant toute utilisation.
- Un objet peut être considérée comme une boîte, décrit par ***un nom*** (significatif), ***un type***(type prédéfini), ***une valeur*** et ***une nature*** (variable, constante ou type(nouveau type à définir)).
- La nature d'objet peut être :
 - Constante : si sa valeur ne change jamais durant l'exécution de l'algorithme ;
 - Variable : si sa valeur est possible de changer durant l'exécution de l'algorithme;
 - Type : nouveau type à définir;

Algorithme

1.1.1 Notion de type prédéfini :

- sont des types simples, à valeur unique. Un type désigne l'ensemble des valeurs permises pour un objet, et l'ensemble des opérateurs autorisés sur les objets de ce type.
 - Les types de base sont : *Entier*, *Réel*, *caractère* et *booléen*.
- a) **Le type Entier** : Les valeurs de ce type sont des nombres entiers positifs, nuls ou négatifs. (ex : 2 , 7 , -74 , 0 , ...).
- b) **Le type Réel** : Le type réel dénote un sous ensemble des nombres réels. (ex : 12 , 7.39 , -4.2 , 3 ,...).
- c) **Le type booléen** : Une valeur booléenne (ou logique) est l'une des deux valeurs '*vrai*' ou '*faux*'. Elle intervient dans l'évaluation d'une condition.

Algorithme

d) Le type caractère: Sa valeur est un caractère quelconque. Un caractère peut appartenir au domaine

- des chiffres de '0' à '9',
- des lettres de 'A' à 'Z' (majuscules ou minuscules)
- des caractères spéciaux ('+' '-' ',' '.' '(' '{' '[' '\$' '%'...).

Un caractère sera toujours noté entre des apostrophes. Le caractère blanc (espace) s'écrit ' '

e) Le type chaîne de caractère : Désigne l'ensemble des chaînes que nous pouvons former par la composition des caractères.

Une chaîne est représentée, aussi, entre apostrophes.

(ex : 'première', 'année', 'licence')

Algorithme

1.1.2. Notion de constante :

- Il arrive parfois que l'on ait besoin d'utiliser des données dont on voudrait qu'elles gardent la même valeur pendant toute la durée d'algorithme. C'est-à-dire qu'une fois déclarée, vous voudriez que votre donnée conserve sa valeur et que personne n'ait le droit de changer ce qu'elle contient.

Ces données particulières sont appelées **constantes**.

- **Syntaxe :** *Const Nom-de-constante = Valeur;*
- **Exemples:** Const PI=3.14;

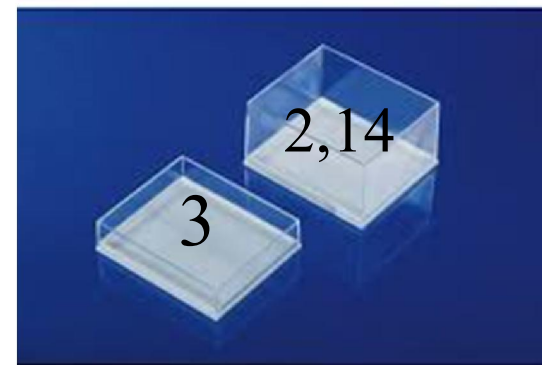
Algorithme

1.1.3. Notion de variable : Une variable c'est une petite information temporaire qu'on stocke dans la RAM. On dit qu'elle est "variable" car c'est une valeur qui peut changer pendant le déroulement de l'algorithme.

- Une variable est constituée de 2 choses :
 - Elle a une **valeur** : c'est le nombre qu'elle stocke, par exemple 5.
 - Elle a un **nom** : c'est ce qui permet de la reconnaître.



Exemple



Algorithme

- Chaque variable doit avoir un nom mais il y a quelques contraintes :
 - Il ne peut y avoir que des lettres minuscules et majuscules et des chiffres (abcABC012...).
 - doit commencer par une lettre.
 - les espaces sont interdits.

Algorithme

➤ *Syntaxe de déclaration d'une variable:*

- Déclarer une variable, c'est de demander à l'ordinateur la permission d'utiliser un peu de mémoire.
- Pour déclarer une variable, il suffit d'indiquer dans l'ordre :
 1. Indiquez le nom que vous voulez donner à la variable
 2. Tapez deux point ‘ : ‘
 3. Le type de la variable que l'on veut créer
 4. Et enfin n'oubliez pas le point-virgule
- **Syntaxe :**

Var Nom-de-variable : Type ;

Algorithme

- **Exemple :**

- Si vous voulez déclarer plusieurs variable de différents types

Var

Compteur : Entier

Lettre : Caractère

- Si vous avez plusieurs variables du même type à déclarer. Il vous suffit de séparer les différents noms de variable par des virgules sur la même ligne :

Var nombre, niveau, age : Entier;

Algorithme

1.2. Partie Actions (instructions) :

- Dans un algorithme, les actions sont généralement décrites par un symbole ou un verbe à l'infinitif choisi pour éviter les confusions. Ce langage est appelé pseudo-code.
- L'instruction c'est une opération élémentaire qui produit un effet spécifié en un temps fini.

Exemples: lire une valeur, afficher un message...

Algorithme

a) Les instructions élémentaires (ou simples) :

a.1) L'Affectation :

- C'est l'action qui consiste à attribuer, une valeur à une variable **V**. par exemple pour affecter '1' au variable **V**, il suffit de noter : **$V \leftarrow 1$** ;
- le type de la valeur doit être compatible avec le type de la variable **V**.

Algorithme

a) Les instructions élémentaires (ou simples) :

a.2) La lecture (saisie) d'une valeur :

- Afin de récupérer et le stocker une valeur donnée par l'utilisateur dans une variable, on va utiliser la fonction '*lire*'.
- Pour demander à l'utilisateur de rentrer une valeur. Vous devez mettre le mot clé '*lire*' suivie par le nom de variable entre parenthèses.
- Syntaxe : *Lire (nom_variable);*
- Par exemple : *lire (age);*

Algorithme

- Lorsque l'algorithme arrive à un *lire*, il se met en pause et attend que l'utilisateur introduit une valeur. Cette valeur sera stockée dans la variable noter entre parenthèses. (ex:"age").
- Il est recommandé de précéder toute instruction de lecture d'une variable par l'affichage d'un message qui invite l'utilisateur à saisir une valeur.

Algorithme

a) Les instructions élémentaires (ou simples) :

a.3) Ecrire (Afficher) un message à l'écran :

- Pour afficher un message à l'écran, une instruction élémentaire, appelée '*écrire*', est disponible.
- Syntaxe : *écrire (" message");* (ex: écrire ("bonjour");)
- Pour afficher la valeur d'une variable avec cette même fonction, on utilise en fait '*écrire*' de la même manière.
- Syntaxe : *écrire (nom_variable);*
- Par exemple : *écrire (age);*

Algorithme

Exercices :

Ecrire l'algorithme qui permet d'afficher le message suivant :

Bonne année

Algorithme

b) Les Expressions élémentaires:

- Une expression est formée d'opérandes et d'opérateurs. Un opérande peut être une constante ou une variable.
- Pour faire une addition, on utilise le signe '+'.
 - Vous devez mettre le résultat de votre calcul dans une variable. On va donc par exemple créer une variable "resultat" de type entier et faire un calcul :

```
resultat : entier ;  
resultat ← 5 + 3;
```

- Bien sûr, rien ne s'affiche. Si vous voulez afficher la valeur de la variable, rajoutez un « écrire » comme suit :

```
ecrire (resultat) ;
```

Algorithme

- Pour les autres opérations, c'est pareil, seul le signe utilisé change :
 - Addition : $+$
 - Soustraction : $-$
 - Multiplication : $*$
 - Division : $/$
 - Modulo : **mod**
- Le modulo est une opération mathématique qui permet d'obtenir **le reste d'une division**.

Algorithme

- Ce qui serait intéressant, est de faire des calculs entre plusieurs variables.

```
resultat ← nombre1 + nombre2;
```

- Cette ligne fait la somme des variables nombre1 et nombre2, et stocke le résultat dans la variable resultat.
- L'évaluation d'une expression repose sur les règles de priorité entre opérateurs selon l'ordre (du plus prioritaire au moins prioritaire) suivant :
 1. Operateur unaires appliqués à un seul opérande : **non**, **-**, **+**
 2. Opérateurs multiplicatifs : *****, **/**, **div**, **mod**, **et**.
 3. Opérateurs additifs : **+**, **-**, **ou**
 4. Opérateurs de relation : **<**, **<=**, **>**, **>=**, **=**, **≠**

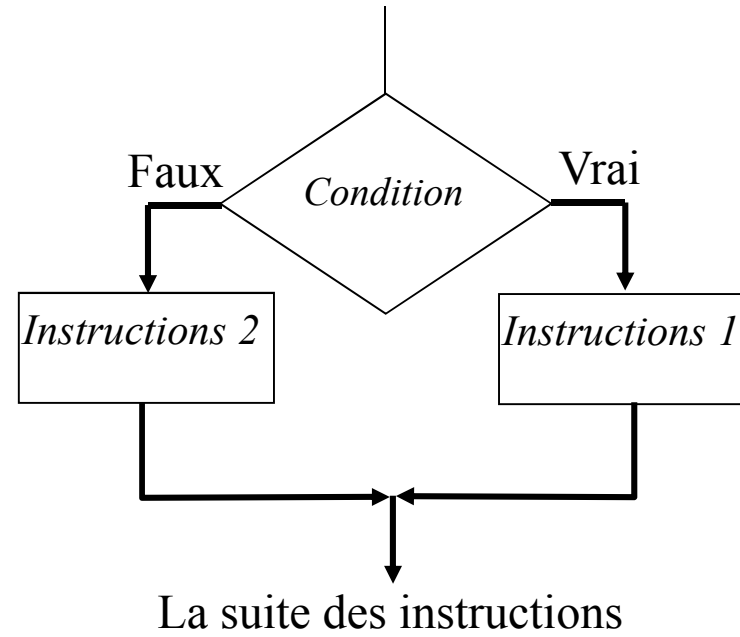
Algorithme

Remarque :

- Pour des opérateurs de même priorité, l'expression est évaluée de gauche à droite.
- S'il y a des parenthèses, on commence par évaluer les plus internes

Exercices :

Ecrire l'algorithme qui permet de calculer la somme de deux nombres entiers A et B introduit par l'utilisateur.



SI, Sinon et Selon

LES STRUCTURES CONDITIONNELLES

Algorithme

c) Structure conditionnelle :

- L'exécution d'un bloc d'instructions peut être conditionnée par la vérification d'une condition.
- Une condition est une expression logique qui ne peut prendre que deux valeurs VRAI ou FAUX.
- Avant de voir comment on écrit une condition, il faut connaître les symboles de comparaison. Ces symboles sont indispensables pour réaliser des conditions.

$A=B$	A est égal à B
$A > B$	A est supérieur à B
$A < B$	A est inférieur à B
$A \geq B$	A est supérieur ou égal à B
$A \leq B$	A est inférieur ou égal à B
$A \neq B$	A est différent de B

Algorithme

c.1) La structure simple :

SI (*condition*) **Alors**
 < *instructions* >
Fsi;

- Si la condition (*condition*) est vérifiée, on exécute les instructions < *instructions* >, et on passe à la suite de l'algorithme. Si la condition n'est pas vérifiée, alors on passe à la suite de l'algorithme.
- **Exemple** : Ecrire l'algorithme qui affiche la valeur absolue d'un nombre entier A.

Algorithme

Algorithme ValAbs;

Var

A : entier

Début

Lire(A);

SI (A < 0) **ALORS**

A $\leftarrow$ A*(-1);

FSI

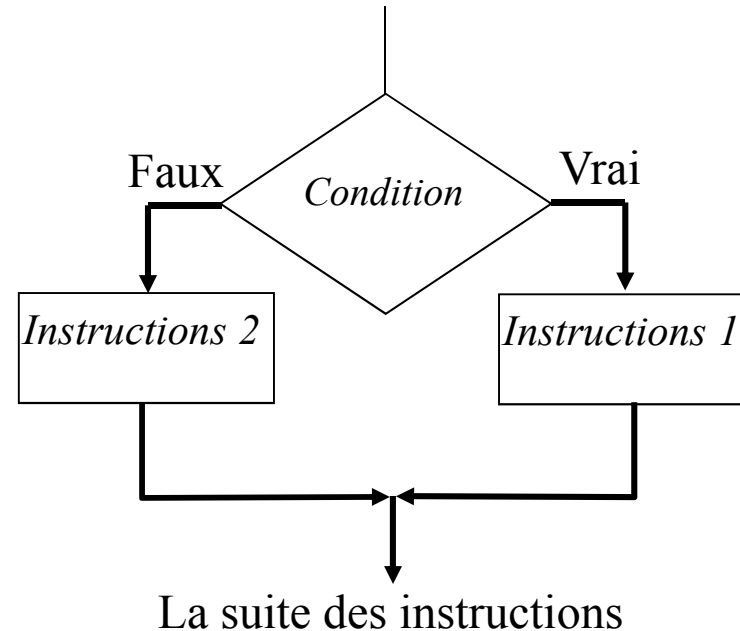
Ecrire ('la valeur absolue est :', A);

Fin.

Algorithme

c.2) La structure complete :

SI (*condition*) **Alors**
< *instructions 1* >
Sinon
< *instructions 2* >
Fsi;



- **Si** la condition (*condition*) est vérifiée, on exécute les instructions < *instructions 1* > et on passe à la suite de l'algorithme; **sinon** on exécute les instructions < *instructions 2* > et on passe à la suite de l'algorithme.

Algorithme

Exemple : Ecrire l'algorithme qui affiche le maximum de deux nombres entiers A et B.

Algorithme Maximum;

Var A, B, Max : entier;

Début

 Lire (A, B);

SI (A ≥ B) **Alors**

 Max ← A

SINON

 Max ← B;

FSI

 Ecrire ('le maximum est :', Max);

Fin

Algorithme

c.3) La structure imbriquée :

```
Si <Condition 1> Alors
    <Instructions1>
Sinon
    Si <Condition 2> Alors
        <Instructions2>
    Sinon
        ....
        ....
        ....
    Fsi ;
Fsi;
```

- Plusieurs conditions peuvent être imbriquées, il s'agit d'un choix multiple.

Algorithme

Exemple : donner un algorithme qui fait la comparaison de deux nombres entiers positifs N1 et N2.

Algorithme Comparaison;

Var

Début

Fin

Algorithme

Exemple : donner un algorithme qui fait la comparaison de deux nombres entiers positifs N1 et N2.

Algorithme Comparaison;

Var N1, N2 : entier;

Début

 Lire (N1, N2) ;

Fin

Algorithme

Exemple : donner un algorithme qui fait la comparaison de deux nombres entiers positifs N1 et N2.

Algorithme Comparaison;

Var N1, N2 : entier;

Début

 Lire (N1, N2) ;

Si (N1= N2) **Alors**

 Ecrire (N1,'=',N2)

Sinon

Si N1>N2 **Alors**

 Ecrire (N1,'>',N2)

Sinon

 Ecrire (N1,'<',N2);

Fsi ;

Fsi ;

Fin

Algorithme

c.4) La structure à choix multiple :

- Toutefois, la structure imbriquée peut nécessiter plusieurs répétitions. Pour éviter ces répétitions dans le test de la valeur d'une seule et même variable, on peut utiliser une autre structure. Cette structure particulière s'appelle "à choix multiple".

Selon <variable de choix>

Choix 1 : Séquence d'instructions 1;

Choix 2 : Séquence d'instructions 2;

.....

.....

Choix n : Séquence d'instructions n;

Fselon;

Algorithme

Exemple : affichage du mois de l'année.

Selon Mois

1 : Ecrire ('Janvier') ;

2 : Ecrire ('Février') ;

.....

12 : Ecrire ('Décembre')

Fselon

Les boucles

LES STRUCTURES ITÉRATIVES

Algorithme

d) Les structures itératives (les boucles) :

- Une boucle est une technique permettant de répéter les mêmes instructions un certain nombre de fois.
- Nous allons voir 3 types de boucles :
 - Pour
 - Tant que
 - Répéter ... jusqu'à

Algorithme

d.1) la boucle « pour » :

- La boucle pour est très utilisée en programmation.
- **Syntaxe:**

```
Pour Compteur ← val_Initiale à val_Finale inc pas=Valeur_de_pas  
    Faire  
        <Bloc d'instructions>  
FPour
```

- **Compteur** est une variable de type entier (ou caractère). Il commence par **val_Initiale** et termine par **val_Finale** .

Algorithme

```
Lire (nb);  
Pour i:= 1 à nb-1  
    Faire  
        ecrire(i);  
Fpour;
```

- **Pas** est un entier qui peut être positif ou négatif. **Pas** peut ne pas être mentionné, car par défaut sa valeur est égal à 1. Dans ce cas, le nombre d'itérations est égal à

$$Val_Finale - Val_Initiale + 1.$$

- **Val_Initiale** et **Val_Finale** peuvent être des valeurs, des variables définies avant le début de la boucle ou des expressions de même type que compteur.

Algorithme

- **Remarque :**

Si la $\text{val_Initiale} < \text{val_Finale}$

Alors c'est une **Incrémentation** du compteur

Sinon c'est une **Décrémentation** du compteur

- **Exemple :**

Calcul de x à la puissance n (x^n) où x est un réel non nul et n un entier positif ou nul.

Algorithme

Algorithme puissance;

Var

Debut

Fin

Algorithme

Algorithme puissance;

Var x, puiss : réel;

n, i : entier;

Debut

Fin

Algorithme

Algorithme puissance;

Var x, puiss : réel;

n, i : entier;

Debut

Ecrire (" Entrez la valeur de x ") ;

Lire (x) ;

Ecrire (" Entrez la valeur de n ") ;

Lire (n) ;

Pour i $\leftarrow$ 1 à n **faire**

puiss $\leftarrow$ puiss*x;

FinPour;

Ecrire (x, " à la puissance ", n, " est égal à ", puiss);

Fin

Algorithme

Algorithme puissance;

Var x, puiss : réel;

n, i : entier;

Debut

Ecrire (" Entrez la valeur de x ") ;

Lire (x) ;

Ecrire (" Entrez la valeur de n ") ;

Lire (n) ;

puiss $\leftarrow$ 1;

Pour i $\leftarrow$ 1 à n **faire**

puiss $\leftarrow$ puiss*x;

FinPour;

Ecrire (puiss);

Fin

Algorithme

d.2) la boucle Tantque :

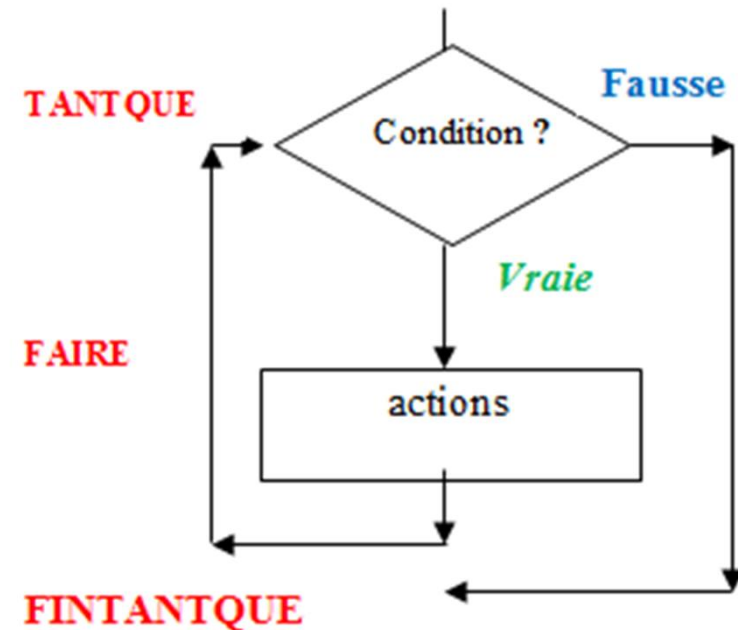
■ Syntaxe :

Tant que (Condition) **faire**

< Instructions >

Ftantque;

- Selon cette boucle on veut dire, Tant que la condition est vraie, on répète l'exécution de bloc < instructions > et dès que la <Condition> devient fausse, on sort de la boucle. Si la condition est initialement fausse, le bloc d'instructions ne sera jamais exécuté.



Algorithme

Exemple : Ecrire un algorithme permettant de calculer la somme des N premiers nombres entiers positifs selon la formule $S=1+2+....+N$.

Algorithme somme;

Var

Debut

Fin

Algorithme

Exemple : Ecrire un algorithme permettant de calculer la somme des N premiers nombres entiers positifs selon la formule $S=1+2+....+N$.

Algorithme somme;

Var I,N,S : entier ;

Debut

Fin

Algorithme

Exemple : Ecrire un algorithme permettant de calculer la somme des N premiers nombres entiers positifs selon la formule $S=1+2+....+N$.

Algorithme somme;

Var I,N,S : entier ;

Debut

 Lire (N) ;

$S \leftarrow 0$;

$I \leftarrow 1$;

Fin

Algorithme

Exemple : Ecrire un algorithme permettant de calculer la somme des N premiers nombres entiers positifs selon la formule $S=1+2+....+N$.

Algorithme somme;

Var I,N,S : entier ;

Debut

 Lire (N) ;

$S \leftarrow 0$;

$I \leftarrow 1$;

Tantque (I <= N) **Faire**

$S \leftarrow S+I$;

$I \leftarrow I+1$;

FTantque;

 Ecrire ('la somme est :',S) ;

Fin

Algorithme

d.3) la boucle Répéter :

■ Syntaxe :

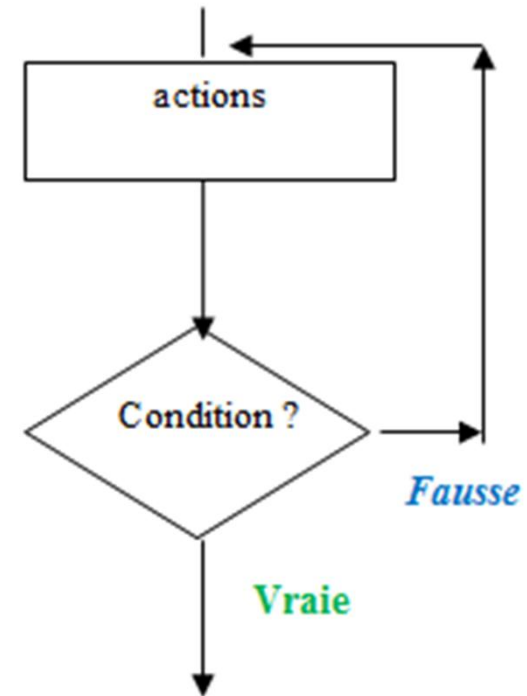
Répéter

< Instructions >

jusqu'à (Condition)

REPETER

JUSQU'A



- Ce type de boucle est très similaire à Tant que, bien qu'un peu moins utilisé en général.
- La seule chose qui change par rapport à Tant que, c'est la position de la condition. Au lieu d'être au début de la boucle, la condition est à la fin.

Algorithme

- la boucle *Tant que* pourrait très bien ne jamais être exécutée si la condition est fausse dès le départ. Par contre la boucle *répéter* s'exécutera au moins une fois.
- Il est donc parfois utile de faire des boucles de ce type, pour s'assurer que l'on rentre au moins une fois dans la boucle.
- **Exemple**

Répéter

Ecrire ('Entrer un nombre positif') ;
Lire(nombre) ;

Jusqu'à (nombre > 0)

Prochaine séance

Les fonctions et les procédures